A Cost-Space Approach to Distributed Query Optimization in Stream Based Overlays

A Cost-Space Approach to Distributed Query Optimization in Stream Based Overlays

Jeffrey Shneidman, Peter Pietzuch, Matt Welsh, Margo Seltzer and Mema Roussopoulos
Division of Engineering and Applied Sciences
Harvard University, Cambridge, MA, USA
hourglass@eecs.harvard.edu

Abstract

Distributed stream-based applications, such as continuous query systems, have network scale *and* time *characteristics that challenge traditional distributed query optimization. The optimization sub-problems of* plan generation *and* service placement *should be integrated to meet these challenges. These tasks have typically been treated as independent sub-problems because of the complexity of their integration. We suggest* cost spaces *as one way to mitigate this complexity. We further consider how cost spaces can be used to allow tractable multi-query optimization.*

1. Introduction

In the beginning, Codd created the relation and the table [1]. And query optimization was without form, and void. But researchers moved to address this void, and said, let there be a query optimizer. And then (when networking came into its own), there were distributed query optimizers [2, 3]. And then followed a long period of relative rest.

This stable equilibrium of distributed query optimization research has been punctuated by recent work in peer to peer databases [4], continuous query systems [5, 6], and other stream-based overlay networks [7]. This paper describes the changes that must occur for distributed query optimization to work well in a general stream-based overlay network (SBON).

An SBON describes an environment where data is streamed from one or more producers to one or more consumers, possibly via a set of services running in-network on additional capable overlay nodes. This general definition is agnostic to data model (relational, semi-structured, etc.) and service model (database operator, application-injected code, etc.). The distributed optimization problem is similar in every case: the end-goal is to satisfy user queries, and when given the choice, to do so in a *good* way with respect to some optimization metric.

The SBON environment presents two challenges: **The first challenge is network *scale*.** Unlike the old database assumption where operator services run at network endpoints, overlay networks permit services to be placed on capable in-network nodes. **The second challenge is *time*.** This challenge has two components. First, whereas a typical database query is finite and short-lived, queries in an SBON can run continuously. Second, node and network characteristics (such as load and latency) are dynamic.

These challenges have significant implications for distributed query optimization. In Section 2, we describe how the two sub-problems of *plan generation* and *service placement* are affected. While the traditional database approach separates these problems to reduce each problem's complexity, it is known that this separation can yield suboptimal decisions [8]. This tradeoff is unwarranted in the SBON setting. In Section 3, we introduce *cost spaces* as a way to reduce the complexity of the optimization subproblems to a point where integrated plan generation and service placement is possible. Cost spaces are metric spaces that allow nodes to express their state by choosing appropriate coordinates. These spaces make the service placement problem tractable and allow plan generation to consider a specific set of nodes. We conclude with open research problems that follow from this integration and call for new research efforts into large scale query optimization.

2. Query Optimization in SBONs

Classic distributed database query optimization has focused on achieving good solutions to three problems:

Data placement considers where to place data so that it may be efficiently queried in the future.

Plan generation creates a logical plan that contains the identity and order of services that must be used to answer a query.

Operator (service) placement considers how to place operators efficiently on a set of physical nodes. We refer to operator placement in an SBON as *service placement* since this processing code may go beyond the confines of a traditional database operator.

Proceedings of the 21st International Conference on Data Engineering (ICDE '05)
1084-4627/05 $20.00 © 2005 IEEE

Some aspects of distributed query optimization are simplified in the SBON setting. Often an SBON is used to relay real-time data from a particular data source to a series of consumers, and no other source can provide this particular data. For instance, live sensor readings from a volcano [9] originate at a particular volcano; one cannot move mountains. Often there is no data placement problem, and we disregard this issue for the remainder of this paper. A second observation is that there is no transaction processing in some stream-based overlay networks. This paper considers systems where data is never changed and re-published.

2.1. Plan Generation

Plan generation takes as input a user query and outputs a logical plan to satisfy that query. The logical plan consists of one or more data endpoints, possibly connected via services, to a consumer. In relational databases, the data may be stored in tables, and example services are `JOIN` and `SELECT`. In these systems, table summary information is used to estimate costs for performing different service orderings. A plan generator selects the least cost plan, which often has the effect of minimizing application response time. Many distributed optimizers use dynamic programming with pruning or some other enumeration algorithm to perform plan selection [8].

The SBON model challenges traditional plan generation in three ways. First, because of long-running stream-based queries, a bad decision in plan generation means that bad plans will cause long-term damage to system capacity and performance. Second, the variable node and network dynamics mean that over the course of a long-running query, an initial plan may become invalid (or suboptimal) and require regeneration[1]. Third, long-running queries increase the likelihood of encountering concurrent plans that can re-use parts of each others' plan trees. This is a double-edged sword: the long-lived nature of queries introduces an optimization opportunity; however, concurrent plans increase the complexity of the optimizer as it must process the union of several plans. This challenge has not been explored in older distributed query optimizers, which "examine one query at a time in isolation and form a plan as if it were the only work running in the system." [11] The long-lived nature of SBON queries, combined with ways of limiting the complexity of this problem (as in Section 3.4), make this challenge more compelling.

1 There has been previous work on regenerating plans due to changing node conditions [10], but our impression is that this has been viewed as an optimization research niche. This niche view is probably correct in a short-lived query, since there is little time to recoup the cost of re-optimization. In a long-running query, recouping costs is less of an issue.

2.2. Service Placement

Service placement takes as input a logical plan and outputs a mapping of each logical service to a physical node in the network. Traditional optimizers vary in how they assign a cost to candidate placement decisions and select the least-cost plan. Kossman gives a good overview of how different models are used in the placement decision [8]. Again, dynamic programming with pruning is often used for placement.

Current service placement algorithms are dramatically affected by the SBON assumptions. The scale challenge is the most obvious change. Distributed databases have previously treated the network as an opaque transport. With the advent of the overlay (e.g., PlanetLab [12]), optimizers are now able to insert application logic into the networking infrastructure. Whereas previous optimizers had a placement choice ranging in the tens of nodes, the next generation overlay-aware optimizers have hundreds or thousands of physical node choices. This is the nail in the coffin for traditional service placement techniques unless there is substantial guidance on where to focus the search.

The next two SBON challenges are similar to those observed in plan generation. First, the changing system dynamics over the course of a long-running query mean that the initial placement may become invalid (or suboptimal) and require regeneration. Second, long-running queries increase the likelihood of being able to merge identical services (serving different queries) into one physical service instance. As in plan generation, there has been little work on dynamic query optimization in databases that takes these changes into account.

2.3. Why Integrate Query Optimization?

Many distributed databases perform plan generation and service placement as a *two-step optimization* [8, page 443]. The idea is to perform plan generation without considering node or network state. Then, immediately before the plan is executed, perform the service placement decision taking into account current network characteristics.

Figure 1 shows an example of an inefficiency that can be caused by separate plan generation and service placement steps, even when optimizing a single query. In this example, the distance between physical nodes corresponds to communication latency. A four-way join operator is decomposed into three two-way joins (Services $S_{1...3}$) and then placed in the SBON. The plan generation phase picks Query Plan 1 for the decomposition, which turns out to be a poorer choice due to the distribution of Producers $P_{1...4}$. Assuming the selectivities of the two plans were roughly the same, Query Plan 2 would have resulted in a more efficient query placement, in that the total data latency is lower. However, this only becomes apparent after examining the network.

Some work in dynamic database query optimization has attempted to blend the two optimization sub-problems. One

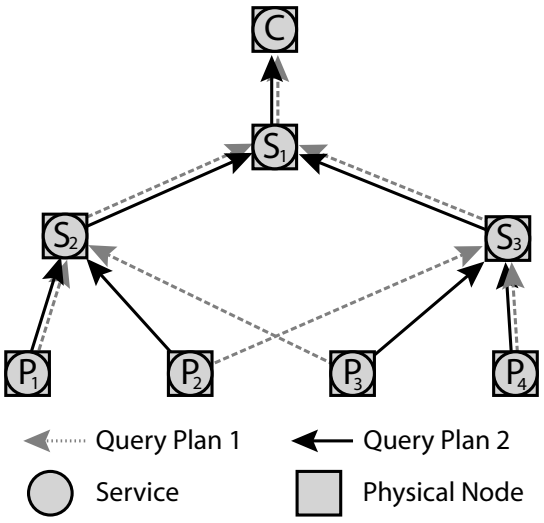

Figure 1. Example of inefficient two-step optimization. *The decomposition of Query Plan* 1 *is less efficient than Plan* 2 *but this becomes apparent only after service placement.*

idea for common queries is to pre-calculate and store plans and sub-plans in the database [13]. At compile time, each plan is generated with a different set of network assumptions. Then, when an expected query is issued, the optimizer examines current network state and tries to find the pre-computed plan that best matches current conditions. This approach is limited in that the optimizer must guess which future node and network states are relevant and worth pre-calculation. Furthermore, it is only applicable to "common" anticipated queries where a plan generation pre-calculation can be performed.

Integration of the two optimization sub-problems would be ideal. Yet, earlier in this section, we observed how dynamic programming in plan generation can be overwhelmed by concurrent queries, and how service placement may be inundated both by concurrent queries and with new choices brought about by increased network scale. We must address these problems in order to proceed with an integration effort.

3. A Cost-Space Approach to Integrated Query Optimization

In this section, we propose a novel approach for an integrated query optimizer, which considers the interdependency of query plan generation and service placement. Our approach is based on the idea of a *cost space*, which captures service placement costs in an efficient way (Section 3.2). The cost of service placement can then be used to guide query plan generation, avoiding consideration of an intractable number of possible query plans (Section 3.3). Finally, pruning within the cost space reduces the complexity of multi-query optimization with a large number of concurrently running queries (Section 3.4).

In the following, we will refer to the instantiation of a query in an SBON as a *circuit*. A circuit can contain *unpinned* services, which are services that can be placed, and *pinned* services, which have a pre-defined network location.

3.1. Cost Spaces

A *cost space* is a multi-dimensional metric space that expresses cost information for service placement decisions. A point in this space corresponds to a physical node, where each coordinate component represents an aspect of the cost of using this node. Costs are either *scalars* or *vectors*. CPU load, memory consumption, and disk capacity are examples of scalar costs because they are properties of a single node, and can be represented in one dimension. Communication latency, communication jitter, and available bandwidth are represented as vector costs because they capture the relationship between this node and other nodes in the network. Vector costs usually require multiple dimensions for accuracy.

A sample cost space (using only vector costs) would be a pure *latency space* [14, 15], where the distance between coordinates is an estimate of communication latency. Even though communication latency on the Internet violates the triangulation inequality, it can be shown that such a metric space can be constructed with only a slight error [16] while using a small number of dimensions. Vector costs be calculated in a distributed and iterative nature by constantly refining the coordinates and correcting for network dynamism [17]. A node calculates its scalar component using a weighting function supplied by the deployer of the cost space. The function is constructed to always be non-negative, where zero represents an ideal value. As a simple example that could be used to capture a node's load, the weighting function could be the squared function as in Figure 2; a node uses the square of its current value as its coordinate in the appropriate dimension.

Cost spaces can be used to express trade-offs between different basic costs. For example, an application may want to create circuits that minimize latency, subject to a CPU load constraint. This example is shown in Figure 2. This graph captures communication latency (x- and y-axes) and CPU load (z-axis). The points in the space are physical nodes in an SBON that is run on top of a simulated transit-stub network topology with 600 nodes. The distance in the x-y plane between two nodes gives an estimate of the communication latency of the two nodes. The height on the z-axis is proportional to the squared CPU load on a node.

The semantics (dimensions, units, and weighting functions) of a particular cost-space must be known by all nodes in the SBON. The SBON can support multiple independent cost spaces, each to suit different classes of applications.

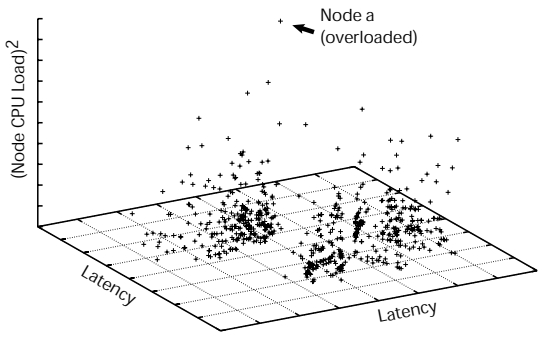

Figure 2. Example of 600 nodes in a 3-dimensional cost space. Communication cost is measured along the x- and y-axes and CPU load along the z-axis. The CPU load dimension uses a squared weighting function to discourage the use of overloaded nodes, such as node a.

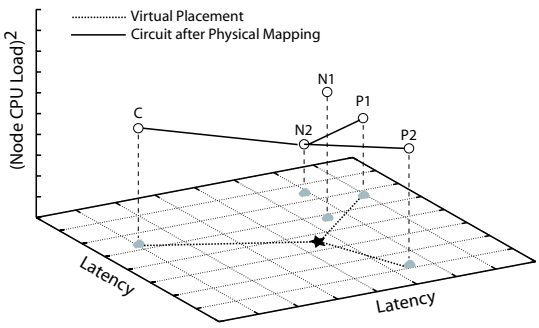

Figure 3. Example of service placement in a cost space. Virtual placement of a single unpinned service is performed in the vector dimensions, and the coordinate marked with the star is chosen. Physical mapping attempts to find the node closest to this starred coordinate, and finds node $N2$. Note that while $N1$ is closer in latency space, its high load makes $N1$ seem far away when the entire cost space coordinate is considered.

In the remainder of this section, we will show how the circuit optimizer can use this cost space for query plan generation and service placement.

3.2. Service Placement

A cost space can be used to efficiently implement service placement in an SBON. Each node in the SBON calculates its own coordinate in the cost space.

The computation is done iteratively to adapt to changes in the system. The goal of circuit optimization is to find a placement of services that minimizes the overall cost of the circuit in the SBON. This physical placement of services is proceeded by two decision phases:

Virtual Placement. A service placement algorithm is used to compute the coordinates of the ideal placement locations for unpinned services in the cost space. Such virtual placement decisions are computationally inexpensive as they do not instantiate services.

An algorithm for scalable, decentralized virtual placement of services in a cost space is *Relaxation placement* [7]. Relaxation placement uses a spring relaxation technique to minimize the costs and approximate optimal placement locations in a latency cost space with respect to global network utilization. It models circuits as springs, such that the spring constant equals the data rate transfered over the link and the spring extension derives from the latency. Services are modeled as massless bodies between springs: Pinned services have a fixed location, whereas unpinned services can move freely. As a result, the function minimized by Relaxation placement is network utilization, expressed as the amount of data in transit in the network. The iterative nature of Relaxation placement allows it to adapt to changing network and circuit conditions. The details of Relaxation placement are described in previous work [7] and are outside the scope of this paper. Other virtual placement algorithms could be based on a centroid calculation or a gradient descent [18] within the cost space.

The virtual placement algorithm operates only over the vector cost dimensions, since the ideal scalar components will all be zero. This is illustrated in Figure 3, which shows placement using the same type of cost space as in Figure 2. Virtual placement is performed in the x-y plane since node load does not affect the placement decision. Scalar dimensions are used in the next phase that performs the physical mapping.

Physical Mapping. Any algorithm that uses a cost space to obtain a good service placement location is faced with a *mapping problem*. The basic problem solved in physical mapping is to find a physical node that is close to the coordinate calculated in the virtual placement. This is a pragmatic interpretation of the idealistic virtual placement; a placement coordinate from the cost space must be mapped back to a physical node before the actual placement of the service can be carried out.

One way to implement a mapping from cost space coordinates to physical nodes is to use a decentralized catalog, such as a distributed hash table (DHT) [19], that returns nodes that are closest to a given coordinate. This requires each node to stores its coordinates in the DHT after transforming its multi-dimensional coordinate to a one-dimensional hash key with a Hilbert curve [20, 21]. Due to the properties of DHT routing [22], a look-up of a coordinate in the DHT then returns the node with the closest ex-

isting coordinate in the system.

The mapping from cost space coordinates to physical nodes introduces a *mapping error* if there are no physical nodes close to a desired coordinate. For example, in Figure 3, the virtual placement chooses the star as the best coordinate for the single unpinned service. Ideally, a physical node with zero load would be present at the star's coordinate. However, the physical mapping finds the closest node to be $N2$, introducing some error. The magnitude of the mapping error depends on the dimensionality of the cost space and the distribution of physical nodes within that cost space. However, experiments have shown that for realistic topologies and latency cost spaces this error remains small [7].

3.3. Plan Generation with Service Placement

As explained in Section 2.3, integrating plan generation and service placement improves the efficiency of query optimization in an SBON. Plan generation and service placement can be integrated in the following manner: When a query is introduced into the system by an application, any node in the network performs a *full circuit optimization*. As in traditional database optimization, a set of candidate plans is created. But in the integrated approach, each plan is virtually placed and physically mapped using the desired cost space. This yields exactly one candidate circuit per plan, with the cost of the circuit representing the current node and network state. The cheapest of these candidate circuits is selected as the circuit and physically placed.

Over time, as network dynamics change, each node that hosts part of a circuit is capable of *re-optimization*. This is a local procedure, where a node can re-run placement and mapping for any service that it hosts. The result may be to migrate the service to a cooperating node so that the best nodes to host a service are consistently used. As part of re-optimization, a node can perform limited plan re-writing as long as it is running all affected services. This could involve the reordering of services, the decomposition of existing services into sub-services to reduce load, or the re-composition of services to reduce network communication.

But it is also possible that a stronger form of re-optimization is required. For instance, the selectivity estimates used to favor one plan over another may change as a circuit matures. In this scenario, a node can trigger the full circuit optimization while the original circuit is still running. If warranted, a new parallel circuit is deployed, cancelling the original less ideal circuit.

Service placement using a cost space provides a technique to reduce the complexity of service placement. By reducing this complexity, a query optimizer can consider the *combined* cost of a query plan and the best service placement for this plan, selecting the plan and placement that have the smallest total cost.

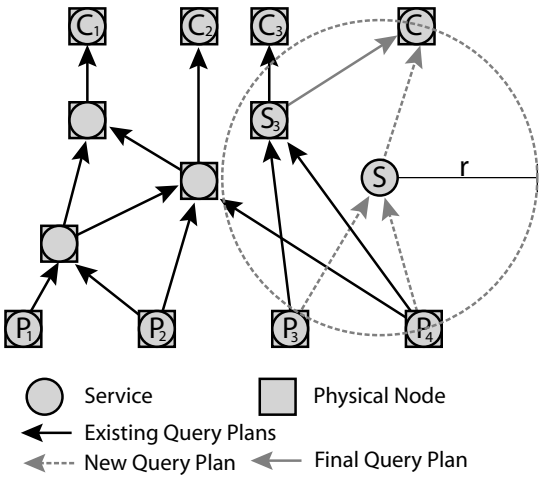

Figure 4. Example of multi-query optimization in a cost space. ***The example shows a multi-query optimizer only considering services within a radius r of a new service in a 2-dimensional latency cost space.***

3.4. Multi-Query Optimization

To perform multi-query optimization, the state space that an integrated query optimizer has to consider is much larger. If there are many concurrent queries in the SBON, a new query can potentially affect any of the existing queries by *reusing* or *transforming* existing services. One way to deal with this enlarged search space of an integrated optimization approach is to use the cost space to prune the search. Standard distributed query optimization techniques can then rewrite the query plans of individual queries and perform multi-query optimization. This idea is based on the observation that query plans that involve operators hosted on physical nodes that are far away in the cost space are less likely to be useful and thus can be ignored by the optimizer. For example, if a circuit only has pinned services in the US, it is unlikely that reusing existing services in Japan will minimize overall cost for the circuit.

When a new circuit is added to the SBON, the cost space can be used for pruning multi-query optimization decisions in different ways. A simple idea is to consider a small *region* in the cost space. The optimizer will then process circuits that fall within this region. For instance, for each unpinned service in a circuit, one implementation could use the Hilbert DHT to look up the closest n nodes that may already be running the same service. This effectively searches around the hyper-sphere surrounding each unpinned service. Other implementations should be possible; this seems like an interesting area for future research.

Figure 4 is an example of an integrated optimizer that performs multi-query optimization for all circuits that fall

within a radius r of a new service S in a 2-dimensional cost space. In this figure, a new circuit with Consumer C is added to the SBON in a 2-dimensional cost space. First, the query optimizer chooses a query plan for the new circuit and calculates the desired placement coordinate for the new Service S in the cost space. Next, it considers multi-query plans involving all circuits that fall within a circle with radius r, namely the circuit with Consumer C_3. Note that the circuits with Consumers C_1 and C_2 are outside of this region of the cost space and are therefore ignored, reducing the complexity of the multi-query optimization decisions. A cheaper query plan and service placement is found by reusing service S_3, which leads to the final query plan that is created in the SBON.

4. Research Challenges

Stream-based overlay networks require the integration of query plan generation and service placement in order to avoid inefficient circuit instantiation. We believe that the abstraction of cost spaces has the potential to enhance large-scale query optimization with new scalable plan generation and service placement techniques. Such techniques are necessary for SBONs because enumeration-based query optimization performs poorly in a large-scale system. Cost spaces help perform targeted pruning in such a way that placement decisions with a high cost are discarded automatically. Multi-query plan optimization can then focus on regions in the cost space that are attractive from a service placement perspective.

However, there are open research challenges that must be addressed. While we have experimented with latency cost spaces and relaxation-based placement in simulation in previous work [7], there is a need to investigate how the dynamic behavior of the network and the data streams will affect circuit optimization in practice. In addition, different, higher-dimensional cost spaces will require the design of novel decentralized implementations of cost spaces and scalable query optimization algorithms that operate within these spaces.

References

[1] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," in *Commun. ACM*, vol. 13, no. 6, 1970.

[2] J. B. Rothnie *et al.*, "Introduction to a System for Distributed DBs (SDD-1)," in *ACM Trans. Database Syst.*, vol. 5, no. 1, 1980.

[3] R. Williams *et al.*, "R*: An Overview of the Architecture," IBM Research Lab, San Jose, Tech. Rep. RJ3325, December 1981.

[4] R. Huebsch, J. M. Hellerstein, N. Lanham, *et al.*, "Querying the Internet with PIER," in *Proc. of VLDB'03*, Berlin, Germany, Sept. 2003.

[5] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases," in *Proc. of SIGMOD'00*, Dallas, TX, 2000.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, *et al.*, "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," in *Proc. of the 1st Biennial Conf. on Innovative Data Systems Research (CIDR'03)*, Asilomar, CA, Jan. 2003.

[7] P. Pietzuch, J. Shneidman, M. Roussopoulos, M. Seltzer, and M. Welsh, "Path Optimization in Stream-Based Overlay Networks," Harvard University, Tech. Rep. TR-26-04, Sept. 2004.

[8] D. Kossmann, "The State of the Art in Distributed Query Processing," *ACM Computing Surveys*, vol. 32, no. 4, Dec. 2000.

[9] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh, "Monitoring Volcanic Eruptions with a Wireless Sensor Network," Harvard University, Tech. Rep. TR–27–04, 2004.

[10] T. Urhan, M. J. Franklin, and L. Amsaleg, "Cost-based Query Scrambling for Initial Delays," in *Proc. of SIGMOD*, June 1998.

[11] M. Stonebraker and J. M. Hellerstein, Eds., *Readings in Database Systems*. Morgan Kaufman, 1998, ch. 4, p. 325.

[12] The Planetlab Consortium, http://www.planetlab.org, 2004.

[13] G. Graefe and K. Ward, "Dynamic Query Evaluation Plans," in *Proc. of ACM SIGMOD'89*. ACM Press, 1989, pp. 358–366.

[14] R. Cox, F. Dabek, F. Kaashoek, *et al.*, "Practical, Distributed Network Coordinates," in *Proc. of the 2nd Workshop on Hot Topics in Networks (HotNets-II)*, Cambridge, MA, Nov. 2003.

[15] M. Pias, J. Crowcroft, S. Wilbur, S. Bhatti, and T. Harris, "Lighthouses for Scalable Distributed Location," in *Proc. of the 2nd Int. Workshop on P2P Systems (IPTPS'03)*, Berkeley, CA, Feb. 2003.

[16] T. E. Ng and H. Zhan, "Predicting Internet Network Distance with Coordinates-Based Approaches," in *Proc. of INFOCOM'02*, June 2002.

[17] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," in *Proc. of the ACM SIGCOMM'04 Conference*, Portland, OR, Aug. 2004.

[18] B. J. Bonfils and P. Bonnet, "Adaptive and Decentralized Operator Placement for In-Network Query Processing," in *Proc. of 2nd Int. Workshop on Info. Proc. in Sensor Networks (IPSN)*, 2003.

[19] I. Stoica, R. Morris, D. Karger, *et al.*, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *Proc. of ACM SIGCOMM'01*, San Diego, CA, Aug. 2001.

[20] H. Sagan, *Space-Filling Curves*. Springer-Verlag, 1994.

[21] A. Andrzejak and Z. Xu, "Scalable, Efficient Range Queries for Grid Information Services," in *Proc. of P2P'02*, 2002.

[22] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *Proc. of Middleware'01*, Heidelberg, Germany, Nov. 2001.